

Spectral algorithms for reaction-diffusion equations

R. V. CRASTER

Imperial College London

and

R. SASSI

Università di Milano, Italy

A collection of codes (in MATLAB & Fortran 77), and examples, for solving reaction-diffusion equations in one and two space dimensions is presented. In areas of the mathematical community spectral methods are used to remove the stiffness associated with the diffusive terms in a reaction-diffusion model allowing explicit high order timestepping to be used. This is particularly valuable for two (and higher) space dimension problems. Our aim here is to provide codes, together with examples, to allow practioners to easily utilize, understand and implement these ideas; we incorporate recent theoretical advances such as exponential time differencing methods and provide timings and error comparisons with other more standard approaches.

The examples are chosen from the literature to illustrate points and queries that naturally arise.

Categories and Subject Descriptors: G. 1.8 [Numerical Analysis]: Partial Differential Equations-Spectral Methods

General Terms: Algorithms

Additional Key Words and Phrases: Fourier transforms, MATLAB, Runge-Kutta methods, Exponential time differencing

1. INTRODUCTION

The aim of this paper is to provide a suite of practically useful and versatile spectral algorithms (in both Matlab and Fortran 77) to efficiently solve, numerically, systems of partial differential equations of the general form:

$$u_t = u_{xx} + u_{yy} + f(u, v), \quad v_t = \epsilon(v_{xx} + v_{yy}) + g(u, v) \quad (1)$$

where $g(u, v)$ and $f(u, v)$ are nonlinear functions of u, v and ϵ a constant; we could also consider them to additionally be functions of the first derivatives of u, v , but we shall not do so here. The methods we describe are applicable to higher dimensions and further coupled equations, however we shall restrain ourselves to consider two space dimensions and two coupled equations. Such equations abound in mathematical biology, ecology, physics and chemistry, and many wonderful mathematical patterns and phenomena exist for special cases. In one space dimension: non-

This work was supported in part by the EPSRC through Grant number GR/S47663/01.

Authors' address: R. V. Craster, Department of Mathematics, Imperial College of Science, Technology and Medicine, London, SW7 2BZ, U.K.; Roberto Sassi, Dipartimento di Tecnologie dell'Informazione, Università di Milano, via Bramante 65, 26013, Crema (CR), Italy (e-mail: sassi@dti.unimi.it)

© 2006 Università degli studi di Milano, Polo didattico e di ricerca di Crema.

Technical Report. Note del Polo, No. 99, 2006.

dimensional models of epidemiology and mathematical biology, such as,

$$u_t = u_{xx} + u(v - \lambda), \quad v_t = \epsilon v_{xx} - uv \quad (2)$$

emerge from the modelling. In this example, there are parameters λ, ϵ , where u, v are the infectives and susceptibles respectively. The parameters λ, ϵ are a removal rate and a diffusivity. These types of equation typically yield travelling waves, [Murray 1993], similar in theory and spirit to that for Fisher's equation which is the usual paradigm. Here issues such as the speed selection for travelling waves, and also accelerating travelling waves are of interest; these are relevant when one species consumes or invades another; [Fisher 1937] was originally interested in the propagation of advantageous genes. Reaction diffusion equations also lead to many other interesting phenomena, such as, pulse splitting and shedding, reactions and competitions in excitable systems, and stability issues. Later, we shall explicitly illustrate the versatility of the scheme presented here versus a wide range of examples from the literature.

Our primary interest is not really in 1D simulations, these are relatively easily undertaken using either method of lines coupled with spatial adaptive schemes [Blom and Zegeling 1994], or finite element collocation schemes [Keast and Muir 1991], or even just simple Crank-Nicholson and finite difference schemes [Sherratt 1997]. Of these, the adaptive schemes seem preferable, in general, since they cluster the grid points in areas of sharp solution gradients. As such the Blom & Zegeling code has been much utilized by one of the authors in a wide variety of different areas ([Balmforth et al. 1999; Balmforth and Craster 2000; Craster and Matar 2000]). The Matlab spectral code we develop, for one dimension, is given in Appendix A, and is certainly competitive with all these schemes and often faster and easier to use.

Unfortunately, in two space dimensions, simulations based upon the more conventional ideas become more time consuming [Pearson 1993; Muratov and Osipov 2001], the latter requiring an hour or so of runtime on an SGI-Cray parallel supercomputer, although the simulations are certainly possible and accurate. It may be that because of this expensive simulation time, comparisons with two-dimensional simulations appear less prevalent in the literature than the 1D cases, even though they are certainly important in modelling pattern creation. However, an idea well-known in the spectral methods community can be used to remove the stiffness often associated with reaction diffusion equations, and thereby allow much larger timesteps to be utilized with an explicit timesolver; consequently the codes run very quickly even on a standard PC or laptop. We utilize Fast Fourier Transforms in space utilizing an integrating factor to remove the stiff terms. The time stepping could be just a standard explicit Runge-Kutta method, and we initially utilize this method. Later we discuss some refinements that adjust standard Runge-Kutta to take into account modifications motivated by the spectral scheme.

Other recent articles on numerically simulating two dimensional reaction-diffusion models utilize wavelet [Cai and Zhang 1998]) or high order finite difference schemes [Liao et al. 2002]; the spectral method could be thought of as the logical extension of finite differences to infinite order, a viewpoint advanced by [Fornberg 1998]. Alternatively physicists and mathematicians interested in the actual processes involved, or underlying mathematical phenomena, use operator splitting methods

[Ramos 2002] (see also Appendix C) or often retreat to finite element simulations ([Tang et al. 1993]) or PDETWO [Melgaard and Sincovec 1981; Davidson et al. 1997] another collocation based scheme. There is only one article that we have found [Jones and O'Brien 1996] that promotes the spectral viewpoint for reaction diffusion equations, and we agree wholeheartedly with its philosophy.

Emboldened by the recent article of [Weideman and Reddy 2001], and the books by [Trefethen 2000], [Boyd 2001], and our experience with using Matlab for other two-dimensional PDE simulations in other contexts ([Balmforth et al. 2004]), we primarily utilize Matlab as our numerical vehicle, although for comparative purposes we also coded the routines in Fortran 77 using Fast Fourier Transform routines from [Fornberg 1998], [Canuto et al. 1988]. Taking advantage of the built-in routines in Matlab, the resulting Matlab codes are extremely concise, typically a page long; an example is given in Appendix A and several are in the accompanying electronic files. Matlab routines are also highly portable between different platforms; our routines require Matlab 5 or higher.

2. FORMULATION

Spectral methods are extremely valuable for generating numerical methods in almost all areas of mathematics. The ability to generate spectrally accurate spatial derivatives means that there is simply no excuse to differentiate poorly. When this is coupled with Fast Fourier transforms and an elegant high-level language such as Matlab it becomes possible to generate versatile and powerful codes. The article [Weideman and Reddy 2001] and book [Trefethen 2000] are quite inspirational in showing the range of what is possible with this combination of tools. We shall present the theory in two space dimensions:

The integrating factor approach that we utilize is to Fourier transform equations (1) to obtain

$$U_t(\omega_x, \omega_y, t) = -(\omega_x^2 + \omega_y^2)U(\omega_x, \omega_y, t) + \mathcal{F}[f(u(x, y, t), v(x, y, t))], \quad (3)$$

$$V_t(\omega_x, \omega_y, t) = -\epsilon(\omega_x^2 + \omega_y^2)V(\omega_x, \omega_y, t) + \mathcal{F}[g(u(x, y, t), v(x, y, t))] \quad (4)$$

where U, V are the double Fourier transforms of u, v , that is,

$$\mathcal{F}[u(x, y, t)] = U(\omega_x, \omega_y, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} u(x, y, t) e^{-i(\omega_x x + \omega_y y)} dx dy. \quad (5)$$

Let us set $\Omega^2 = \omega_x^2 + \omega_y^2$, and explicitly remove the linear pieces of the transformed equations using integrating factors, setting:

$$U = e^{-\Omega^2 t} \tilde{U}, \quad V = e^{-\epsilon \Omega^2 t} \tilde{V}, \quad (6)$$

such that now

$$\partial_t \tilde{U} = e^{\Omega^2 t} \mathcal{F}[f(u, v)], \quad \partial_t \tilde{V} = e^{\epsilon \Omega^2 t} \mathcal{F}[g(u, v)]. \quad (7)$$

At this point, in practical terms, we discretize the spatial domain, considering N_x and N_y equispaced points in the x and y directions. Then we utilize the discrete FFT so equation (7) becomes a system of ODEs parameterized by the Fourier modes (distinguished by a couple of indices ij) so

$$\partial_t \tilde{U}_{ij} = e^{\Omega_{ij}^2 t} \mathcal{F}[f(u_{ij}, v_{ij})], \quad \partial_t \tilde{V}_{ij} = e^{\epsilon \Omega_{ij}^2 t} \mathcal{F}[g(u_{ij}, v_{ij})], \quad (8)$$

where $u_{ij} = u(x_i, y_j)$, $v_{ij} = v(x_i, y_j)$ and $\Omega_{ij}^2 = \omega_x^2(i) + \omega_y^2(j)$. Periodic boundary conditions are implicitly set at the extremes of the spatial domain. Henceforth we will suppress the ij indices and consider this discretization understood. In practice one takes, say, $N_x = N_y = N = 128$ and utilize 128×128 Fourier modes.

The spatial derivatives have now disappeared, along with the stiffness that they had introduced, and the resulting ODEs are now simple to solve with an explicit solver, say, Runge-Kutta. There are issues that arise at this stage, for instance, the fixed points of these equations are different from those of the original ones, aliasing could be an issue and, if we use Runge-Kutta or some other scheme, how does the local truncation error depend upon Ω_{ij} ? We leave these issues aside until later in the article.

Adopting the standard notation for order M Runge-Kutta methods, with time step Δt , that is to advance from $t_n = n\Delta t$ to $t_{n+1} = (n+1)\Delta t$ for an ODE $y_t = f(t, y)$: $y_{n+1} = y_n + \sum_{i=1}^M c_i k_i$ and each k_i is $k_i = \Delta t f(t_n + a_i \Delta t, y_n + \sum_{j=1}^{i-1} b_{ij} k_j)$. The a_i, c_i, b_{ij} are given by the appropriate Butcher array; an almost infinite number of different schemes exist. Two popular ones are given in Numerical Recipes [Press et al. 1992]: classical fourth order and the Cash-Karp embedded scheme [Cash and Karp 1990], we utilize these in our numerics.

For our purposes we apply the general explicit Runge-Kutta formula to the equations (8) for \tilde{U}_{ij} and \tilde{V}_{ij} . Notationally, we denote μ_i and ν_i to be the k 's associated with the \tilde{U} and \tilde{V} equations respectively. The right-hand sides have slightly unsettling exponential terms in t and it is convenient to set replacement variables as

$$\tilde{\mu}_i = \mu_i e^{-\Omega^2 t_n}, \quad \tilde{\nu}_i = \nu_i e^{-\epsilon \Omega^2 t_n}. \quad (9)$$

We write the formulae out for U and V since it is simpler to just work with the transforms of the physical variables rather than the physical variables themselves. The upshot is that with an M -stage Runge-Kutta scheme,

$$U_{n+1} = e^{-\Omega^2 \Delta t} \left[U_n + \sum_{i=1}^M c_i \tilde{\mu}_i \right], \quad V_{n+1} = e^{-\epsilon \Omega^2 \Delta t} \left[V_n + \sum_{i=1}^M c_i \tilde{\nu}_i \right], \quad (10)$$

where the modified $\tilde{\mu}_i$ and $\tilde{\nu}_i$ terms are

$$\begin{aligned} \tilde{\mu}_i &= e^{\Omega^2 a_i \Delta t} \Delta t \mathcal{F} \left\{ f \left[\mathcal{F}^{-1}(U_{n+a_i}), \mathcal{F}^{-1}(V_{n+a_i}) \right] \right\}, \\ \tilde{\nu}_i &= e^{\epsilon \Omega^2 a_i \Delta t} \Delta t \mathcal{F} \left\{ g \left[\mathcal{F}^{-1}(U_{n+a_i}), \mathcal{F}^{-1}(V_{n+a_i}) \right] \right\}, \end{aligned} \quad (11)$$

and the values of U and V at the intermediate steps are

$$U_{n+a_i} = e^{-\Omega^2 a_i \Delta t} \left[U_n + \sum_{j=1}^{i-1} b_{ij} \tilde{\mu}_j \right], \quad V_{n+a_i} = e^{-\epsilon \Omega^2 a_i \Delta t} \left[V_n + \sum_{j=1}^{i-1} b_{ij} \tilde{\nu}_j \right]. \quad (12)$$

That is, one works entirely in the spectral domain and one inverts a transform to recover u and v . Clearly a fair amount of Fourier transforming to and from, is involved and this is the primary numerical cost. Fortunately Matlab has simple to use multi-dimensional Fast Fourier Transform (FFT) routines, (`fft`, `ifft`, `fft2`, `ifft2`) and many routines are available in Fortran (or C).

The essential point is that by removing the stiffness one can use explicit high-order timesolvers and rapidly and accurately move forwards in time, this is vastly superior to using implicit schemes particularly in higher dimensions (see Appendix C). There are some slight deficiencies in the method that can be removed using more recent ideas and we shall return to the theory in section 3.3.2.

3. ILLUSTRATIVE EXAMPLES

We choose a range of illustrative examples that are of current and recurring interest, and which cover pitfalls and natural questions that arise.

3.1 One dimensional models

3.1.1 *Fisher's equation: Speed selection.* Fisher's equation

$$u_t = u_{xx} + u(1 - u), \quad |x| < L \quad (13)$$

provides a nice demonstration; there is a detail that is worth investigating: the speed selection associated with exponential decay of the initial condition. Numerically this has been an issue for other approaches such as moving mesh schemes ([Qiu and Sloan 1998]) with some authors recommending that such schemes be used with caution upon this type of problem ([Li et al. 1998]).

We utilize an initial condition

$$u(x, 0) = \frac{1}{2 \cosh \delta x}$$

that has exponential decay $\exp(-\delta|x|)$ as $|x| \rightarrow \infty$. Theoretically, one expects travelling waves to develop from such an initial condition on an infinite domain, which we truncate at some large, but finite, value, say, $L \sim 150$, and what is particularly interesting is that the system then selects the constant velocity at which the developed fronts propagate, c , and the velocity is a function of the decay rate of the initial condition:

$$c = 2 \quad \text{for } \delta > 1, \quad c = \delta + \frac{1}{\delta} \quad \text{otherwise.} \quad (14)$$

We easily extract the velocity from the simulations, figure 1, and evidently we recover these theoretical values. It is essential that L is taken to be large enough that the initial condition is effectively zero at L , here $L \sim 150$. The figure shows the front position, $X(t)$, here taken as the point where $u(X, t) = 10^{-4}$, versus time, the slope gives the velocity; for convenience ct is also shown, note the offset in panel (b) is not relevant as it is the slope that concerns us.

The simulations in Fortran 77 are performed virtually instantaneously, with the Matlab simulations taking a few seconds.

3.1.2 *Gray-Scott: Pulse splitting.* Another very interesting feature of many reaction diffusion equations is pulse splitting or shedding; a propagating pulse is unstable, and the unstable eigensolutions lag behind the pulse causing a daughter pulse to break off. This is particularly pronounced in the Gray-Scott equations, fortunately they have a pleasant and simple nonlinearity in the reaction terms that makes them amenable to analytical approaches [Doelman et al. 1997; Reynolds

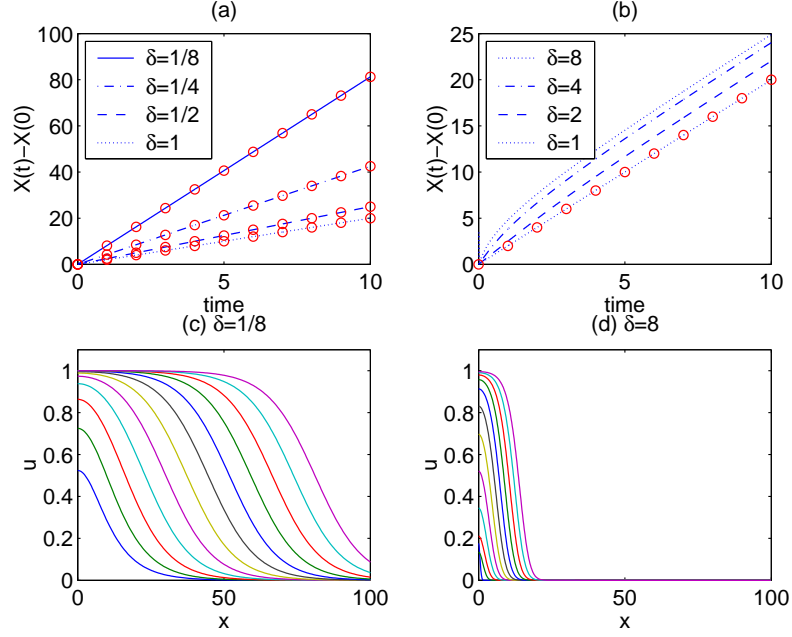


Fig. 1. Panels (a) and (b) show front positions versus time for various values of δ . In panel (a) these are shown for $\delta \leq 1$. Panel (b) shows front positions for $\delta \geq 1$. Circles show ct , c taken from (14). Panels (c) and (d) show profiles of u at unit time intervals until $t = 10$.

et al. 1997]. They also form a tough test upon any numerical scheme as the splitting events and subsequent structure must be captured correctly both in space and time.

The equations (v is the activator and u the inhibitor) are:

$$u_t = u_{xx} - uv^2 + A(1 - u), \quad v_t = \epsilon v_{xx} + uv^2 - Bv. \quad (15)$$

A couple of illustrative plots are given in figure 2, these have initial conditions

$$u = 1 - \frac{1}{2} \sin^{100}(\pi(x - L)/2L), \quad v = \frac{1}{4} \sin^{100}(\pi(y - L)/2L)$$

where we choose the half domain length, L , to be 50. These are chosen to replicate a figure from [Doelman et al. 1997]; notably the simulations differ as the boundary conditions here at $\pm L$ are periodic, and eventually a steady spatially periodic state emerges. The simulations in [Doelman et al. 1997], and reproduced in panel (b) of figure 2, utilize Dirichlet, that is, fixed values of u, v , conditions and hence there is a minor discrepancy close to the edges of the domain, this is most noticeable in u . The Matlab file for this computation is given in Appendix A, and it takes 12 seconds to run on a 1GHz Pentium 3 Dell Laptop running Linux. Comparative computation times are probably meaningless as computational power will ever increase, our only point being that these computations are relatively fast versus competitors even with modest computing facilities available to all/ most undergraduates. The adaptive scheme takes a few minutes depending upon the number of grid points utilized,

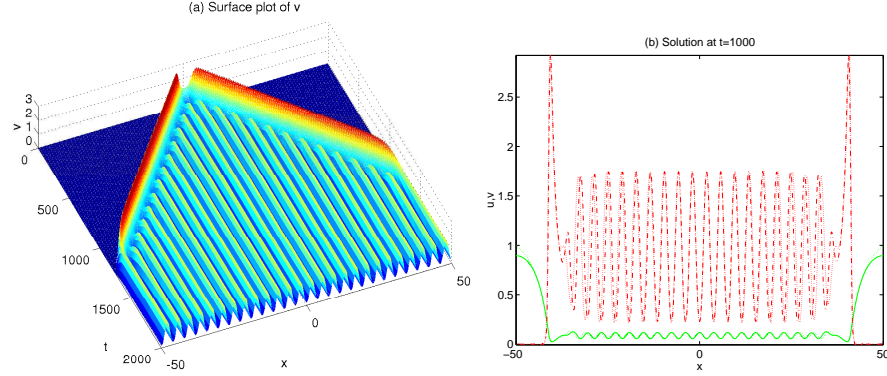


Fig. 2. Panel (a) shows the outward propagating pulses and the shedding phenomena for v out to $t = 2000$. Panel (b) shows both u (solid) and v (dot-dashed) at $t = 1000$, the dotted lines also shown come from the adaptive scheme [Blom & Zegeling 1994]. The parameters chosen here are $a = 9, b = 0.4, \epsilon = 0.01$ where $A = \epsilon a, B = \epsilon^{1/3} b$.

typically 500 points.

3.1.3 Autocatalysis: Oscillatory fronts. Many reaction-diffusion equations arise in combustion theory, or in related chemical models. One such model, in non-dimensional terms, is

$$u_t = u_{xx} + v f(u), \quad v_t = \epsilon v_{xx} - v f(u), \quad (16)$$

where

$$f(u) = \begin{cases} u^m, & u \geq 0, \\ 0, & u < 0 \end{cases} \quad (17)$$

and ϵ is the inverse of the Lewis number (the ratio of diffusion rates). It arises when two chemical species U and V react such that $mU + V \rightarrow (m+1)U$; the two species have different diffusivities, their ratio being ϵ ($\epsilon < 1$ is the regime of interest). What is particularly interesting in this model is that steady travelling waves occur for low values of m , their speed is a function of m, ϵ and the fronts steepen dramatically for large m . In fact, in full nonlinear simulations as m increases a Hopf bifurcation occurs, and as it increases yet further one gets chaotic behaviour at the wave front. This behaviour is detailed in [Balmforth et al. 1999], [Metcalf et al. 1994] and similar features arise in combustion models, see for instance [Bayliss et al. 1989] where $f(u)$ is replaced by exponential Arrhenius reaction terms. For our verification purposes we compared with the computations of [Balmforth et al. 1999], and initiated the computations with

$$u = \frac{1}{2} (1 + \tanh(10(10 - |y|))), \quad v = 1 - \frac{1}{4} (1 + \tanh(10(10 - |y|)))$$

that is, a sharp localized disturbance and obtained perfect agreement even for extremely steep fronts. Figure 3 shows typical results with periodic fluctuations at $\epsilon = 0.1$ and $m = 9$ with more extreme behaviour with $m = 11$. It is notable that all the delicate behaviour, rocking fronts and transitions to apparently chaotic

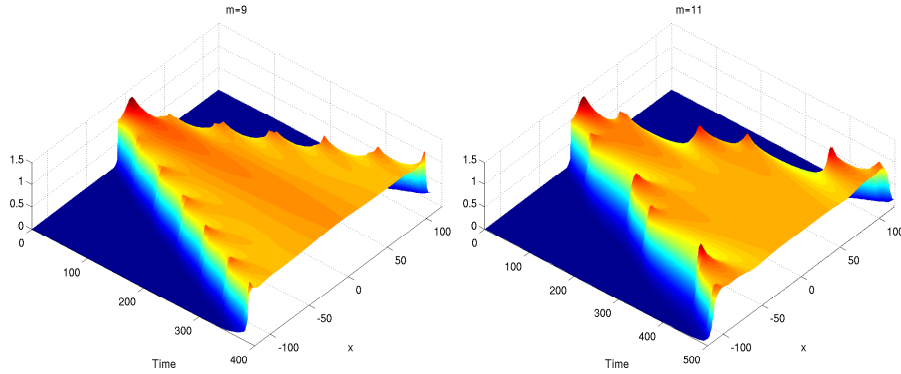


Fig. 3. Typical results for the autocatalytic model with $\epsilon = 0.1$ and $m = 9$ and $m = 11$.

behaviour, is accurately captured, together with the very steep fronts; this is a challenge for any numerical scheme, a minor aside is that our Fortran 77 code had difficulties with this computation until we imposed symmetry conditions at the end of every timestep, or zeroed the imaginary part of the inverted result.

This leads us to an algorithmic detail: in the Matlab codes we use separate variables u and v and transform, and invert, each independently and just use the real parts of each inverse - automatically zeroing the imaginary parts and thereby preventing rounding errors mounting up over time. However, this is actually a bit wasteful as one could combine u and v to be the real and imaginary parts of a single transform variable and just halve the amount of work; the Fortran codes use the latter approach. This only seems to cause problems in this particular example as the extreme powers 11 magnify rounding errors and, for the Fortran code, the minor modification described above is required to maintain accuracy.

3.2 Two dimensional examples

It is in higher dimensions that the ideas presented here really become of serious value. We choose to illustrate the numerical algorithms using a couple of non-trivial examples from the reaction-diffusion equation literature. Stable, labyrinthine patterns arising in FitzHugh-Nagumo type reaction diffusion equations [Hagberg and Meron 1994], and pulse splitting from the Gray-Scott equations [Muratov and Osipov 2001].

3.2.1 Gray-Scott. If we have radial symmetry then many of the 1D schemes can be utilized in radial coordinates, an adaptive code [Blom and Zegeling 1994] is particularly convenient as it takes advantage of the [Skeel and Berzins 1990] discretization that automatically incorporates the coordinate singularity. We can therefore check the numerical simulations of the 2D spectral code versus these. The Gray-Scott model has delicate features that are not easy for a numerical scheme to extract, and it is susceptible to small perturbations generating instabilities.

As noted earlier, the Gray-Scott equations have provided an interesting test bed for theoreticians exploring pulse splitting and so-called auto-solitons and their stability [Doelman et al. 1997; Reynolds et al. 1997; Muratov and Osipov 2001; 2002];

Technical Report. Note del Polo, No. 99, 2006.

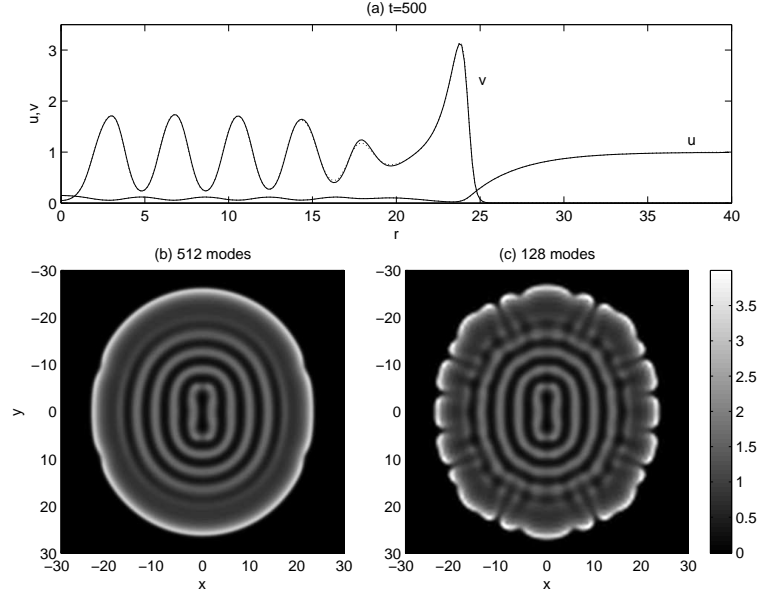


Fig. 4. Panel (a) shows the outward propagating rim and the shedding phenomena for u, v , at $t = 500$ for an axis-symmetric initial condition, along a radial line; the barely visible dotted lines also shown come from the adaptive scheme [Blom & Zegeling 1994]. Panels (b) and (c) shows a planview of v at $t = 500$ starting from non-axisymmetric initial data, the two panels show 512 and 128 modes respectively (same parameters as figure 2). For a better display, resolution in panel (c) was improved using Fourier cardinal function interpolation (see Appendix B).

it is also notable that the model also arises in biological contexts [Davidson et al. 1997]. Different authors prefer different rescalings according to the physics/biology that they wish to emphasise, we shall not enter that debate here. The equations we use are

$$u_t = u_{xx} + u_{yy} - uv^2 + A(1 - u), \quad v_t = \epsilon[v_{xx} + v_{yy}] + uv^2 - Bv. \quad (18)$$

The two-dimensional analogue of the pulse-splitting events of figure 2 are shown in figure 4; using axisymmetric initial conditions for panel (a):

$$u = 1 - \frac{1}{2} \exp(-r^2/20), \quad v = \frac{1}{4} \exp(-r^2/20)$$

where $r^2 = x^2 + y^2$, allows us to verify the two-dimensional computations in a non-trivial way as the system is highly unstable. Although not shown, it is particularly striking how perfect axisymmetry is retained by these axisymmetric computations. Altering the initial conditions to break the axisymmetry to the above, but with $r^2 = x^2/2 + y^2$ leads to the oval pattern of alternating high and low concentrations shown in figure 2 panels (b) and (c). What is particularly notable is that using fewer modes leads to an attractive, but evidently erroneous, pattern.

3.2.2 Labyrinthine Patterns. A striking and interesting group of patterns that emerge in models of catalytic reactions are growing labyrinthine patterns [Hagberg

and Meron 1994; Meron et al. 2001]. Starting from a non-axisymmetric initial condition strongly curved portions move more rapidly and the pattern lengthens. Regions of high concentrations repel and, hence from the periodicity of the domain, the patterns turn inward until an equilibrium is reached. An illustrative simulation is given in figure 5. The computation utilizes 128×128 Fourier modes on a 200×200 grid. Doubling the number of Fourier modes makes no discernable difference; more detailed numerical error discussions are in a later section.

The governing equations are that

$$u_t = u - u^3 - v + \nabla^2 u, \quad v_t = \delta(u - a_1 v - a_0) + \epsilon \nabla^2 v \quad (19)$$

where u, v represent activator and inhibitors. The parameters $a_0, a_1, \epsilon, \delta$ lead one from one regime to another, see [Hagberg and Meron 1994] for details. We begin the simulation in figure 5 from initial conditions

$$u = a_1 v_- + a_0 - 4a_1 v_- e^{-0.1(x^2 + 0.01y^2)}, \quad v = v_- - 2v_- e^{-0.1(x^2 + 0.01y^2)}. \quad (20)$$

This being an elliptical mound of chemical concentrations of sufficient magnitude to trigger the reaction. Here $v_- = (u_- - a_0)/a_1$ is found from, u_- , which is the smallest real root of the cubic $a_1 u^3 + u(1 - a_1) - a_0 = 0$. This (u_-, v_-) state being stable. As one notes from the figure, the evolution is non-trivial and the edges of the concentrations are sharp and steep, all features that test the robustness of the scheme. Verification follows from comparison with the adaptive scheme for axisymmetry, and since a stationary state emerges one can also generate the final shape from a boundary value problem; all computations agree. Figure 5 only shows the evolution of u , that of v is qualitatively similar.

It is worth noting that other behaviours are possible for these equations in other parameter regimes than those chosen here.

3.3 Refinements

All of the figures shown are generated using a fixed time step, and the classical standard fourth order Runge-Kutta scheme; this was 0.1 in all cases except the autocatalytic problem with large m where we took a timestep of 0.02. This is deliberate to demonstrate that sophisticated algorithms are not vital, however it is not pleasant to have no error control or indeed no idea of how accurate the solution actually is at each time step. An overly enthusiastically large choice for the time step could lead to numerical instabilities and accumulated error. The results we present are all generated using Matlab; the Matlab codes that we present in the appendices are efficient as teaching tools, and, to a certain extent as research tools; the high level language gives short and easily understandable code. However, traditional languages such as Fortran or C, C++ will often run much faster, at least versus interpreted Matlab code, and to complement the Matlab codes we also provide the source codes in Fortran 77.

3.3.1 Adaptive time stepping. Adaptive time stepping based upon embedded 5th order Runge-Kutta schemes in the usual manner, see for instance [Cash and Karp 1990; Press et al. 1992], is easily implemented and we do so. The user can replace these weights with their favourite scheme [Dormand and Prince 1980], say, but this will change little in practice. These codes incorporate an error tolerance

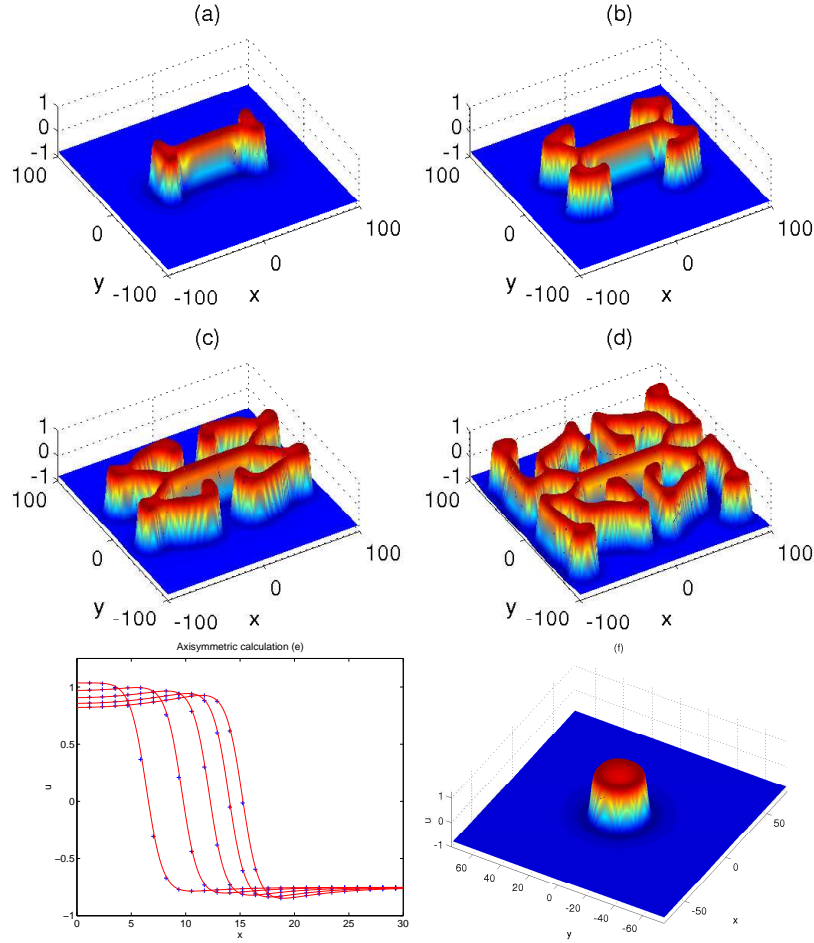


Fig. 5. Panels (a) to (d) show the emerging labyrinthine pattern for u at times $t = 200, 400, 600, 1000$. Panels (e) and (f) an axisymmetric computation, same initial conditions as equation (20) bar that $0.01y^2 \rightarrow y^2$, at $t = 10, 20, 30, 40, 50$, in (e) the solid lines come from the adaptive scheme of [Blom & Zegeling 1994] and crosses from the spectral method. The parameters chosen here are $a_0 = -0.1, a_1 = 2, \epsilon = 0.05, \delta = 4$.

and utilize local extrapolation, these codes often settle to using surprisingly large time steps $dt \sim 0.5$ or larger for even moderate error tolerances (relative error $\sim 10^{-4}$) and this further speeds the computations.

3.3.2 Exponential time differencing. Integrating factor ideas are not new, there are actually several non-appealing aspects of the approach. On the basis of “truth in advertising” we must reveal them. A philosophically unpleasant feature is that one notes that the fixed points of equations (7) are not the same as those of the original untampered equations (3), (4). We are unaware of any circumstance in which this has led to any problems, but it is not nice. A more serious fact that counts in its

disfavour is that the local truncation error, when $\Omega^2 \Delta t \ll 1$, for the time stepping schemes we use are, for say 4th order Runge-Kutta, $O(\Omega^2 \Delta t)^5$. This is apparently disastrous as Ω can be large and we now have fourth order accuracy in time, but with a large pre-multiplicative factor. However, it is important to recall that the local truncation error involves expanding $\exp(-\Omega^2 \Delta t)$ terms which are, practically, exponentially small for large Ω relative to Δt . Nonetheless we are clearly picking up an additional contribution to the numerical error for moderate values of Ω . This is surmountable, basically one designs a time stepping scheme that correctly incorporates the exponential behaviour, a recent article, [Cox and Matthews 2002], derives several exponential Runge-Kutta schemes; this is an active area of current research with modifications of their scheme by [Kassam and Trefethen 2003] to overcome a numerical instability and by [Krogstad 2005] generating a scheme with smaller local truncation error and better stability properties. Krogstad also notes the very interesting link with the commutator-free Lie group methods of [Munthe-Kaas 1999] and undoubtedly this area will develop further.

In essence the exponential time differencing idea applied here for U , involves utilizing the integrating factor $\exp(\Omega^2 t)$ and multiplying equation (3) through by it and then we integrate over a time step to obtain:

$$U_{n+1} = U_n e^{L\Delta t} + e^{L\Delta t} \int_0^{\Delta t} e^{-L\tau} N_u(u(t_n + \tau), v(t_n + \tau), t_n + \tau) d\tau$$

where we have rewritten equation (3) as

$$U_t = Lu + N_u(u, v)$$

that is, with a Linear piece (here $-\Omega^2$) and a Nonlinear piece (here the Fourier transform of the nonlinear reaction terms); this is the notation used in the relevant literature. The interesting departure, and distinguishing feature, from the standard integrating factor method is that one then approximates the integral and the truncation error is then independent of Ω^2 . The article by [Cox and Matthews 2002] contains various approximations to the integral and numerical comparisons of methods.

It is important to bring these “state-of-the-art” solvers into the more applied domain and enable other researchers to take advantage of them. We utilize the fourth order Runge-Kutta-like scheme of Krogstad,

$$\begin{aligned} U_{n+1} = & e^{L\Delta t} U_n + \Delta t [4\phi_2(L\Delta t) - 3\phi_1(L\Delta t) + \phi_0(L\Delta t)] N_u(U_n, V_n, t_n) + \\ & 2\Delta t [\phi_1(L\Delta t) - 2\phi_2(L\Delta t)] N_u(\mu_2, \nu_2, t_n + \Delta t/2) + \\ & 2\Delta t [\phi_1(L\Delta t) - 2\phi_2(L\Delta t)] N_u(\mu_3, \nu_3, t_n + \Delta t/2) + \\ & \Delta t [4\phi_2(L\Delta t) - \phi_1(L\Delta t)] N_u(\mu_4, \nu_4, t_n + \Delta t) \end{aligned}$$

with the stages μ_i as

$$\mu_2 = e^{L\Delta t/2} U_n + (\Delta t/2) \phi_0(L\Delta t/2) N_u(U_n, V_n, t_n)$$

$$\begin{aligned} \mu_3 = & e^{L\Delta t/2} U_n + (\Delta t/2) [\phi_0(L\Delta t/2) - 2\phi_1(L\Delta t/2)] N_u(U_n, V_n, t_n) + \\ & \Delta t \phi_1(L\Delta t/2) N_u(\mu_2, \nu_2, t_n + \Delta t/2) \end{aligned}$$

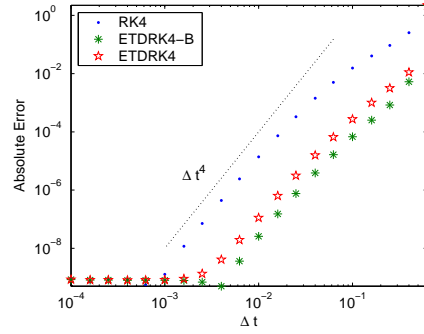


Fig. 6. The 1-D Gray-Scott equation (15) is solved using different time steps Δt with the same parameters values of figure (2). The final solution obtained at $t = 200$ is compared, for each method (RK4, ETDRK4 and ETDRK4-B), with a gold-standard run (computed with ETDRK4-B and $\Delta t = 10^{-5}$); the maximum absolute errors are displayed as a function of the time step.

$$\begin{aligned} \mu_4 = & e^{L\Delta t}U_n + \Delta t [\phi_0(L\Delta t) - 2\phi_1(L\Delta t)] N_u(U_n, V_n, t_n) + \\ & 2\Delta t \phi_1(L\Delta t) N_u(\mu_3, \nu_3, t_n + \Delta t). \end{aligned}$$

The functions ϕ_i are defined as

$$\phi_0(z) = \frac{e^z - 1}{z}, \quad \phi_1(z) = \frac{e^z - 1 - z}{z^2}, \quad \phi_2(z) = \frac{e^z - 1 - z - z^2/2}{z^3}$$

and these are precisely the terms that emerge naturally in the Lie group methods [Munthe-Kaas 1999]. The original Cox & Matthews scheme involves a split-step and has marginally worse error and stability properties. There are also slight problems associated with capturing the behaviour of $\phi_i(z)$ uniformly as $z \rightarrow 0$ and [Kassam and Trefethen 2003] suggest a remedy; one uses an integral in the complex plane [Higham 1996], and we also use this approach in our algorithms. For brevity we have presented the scheme for U alone, the V equations follow in a similar fashion.

We label this as a fourth order Exponential Time Differencing Runge-Kutta (ETDRK4-B) scheme to distinguish it from a standard Runge-Kutta scheme and to be consistent with the notation of [Cox and Matthews 2002; Kassam and Trefethen 2003; Krogstad 2005]. It is also worth noting that various other exponential Runge-Kutta schemes have been developed by other authors, see [Vanden Berghe et al. 2000] and the references therein to overcome this difficulty arising in other contexts.

A reasonably large numerical overhead is involved in setting up the fourth-order scheme using the device suggested by [Kassam and Trefethen 2003]; if one utilizes an adaptive scheme in time then this overhead must be regularly recomputed and this then becomes expensive, hence we do not adapt the ETDRK4 schemes in time.

A numerical 1-D comparison of the ETDRK4 scheme of ([Cox and Matthews 2002]), the improved ETDRK4-B ([Krogstad 2005]), and the more standard RK4 scheme is shown in figure (6). It is noticeable that ETDRK4-B proved to give the smaller error; both it and the ETDRK4 scheme provide an order of magnitude improvement over the RK4 scheme for larger timesteps, rewarding the extra programming effort. The errors all scale with the expected Δt^4 scaling.

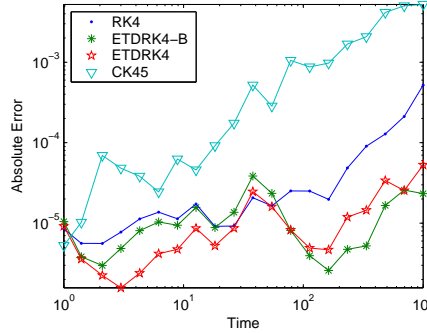


Fig. 7. Absolute errors of the computational methods with respect to a gold-standard run obtained with ETD4RK4-B using $\Delta t = 0.01$. The 2-D equation being solved is (19), which leads to labyrinthine patterns. Most schemes (RK4, ETD4RK4, ETD4RK4-B) use a fixed time step of $\Delta t = 0.1$, CK45 adapts its time step to contain local absolute error below 10^{-4} , leading to an average $\Delta t \approx 0.62$. The smaller computational time (about 11.6 against 22.7 minutes) is paid out with a larger error. Note that Matlab is not very efficient when it comes to loops; with Fortran the difference in execution times is wider (2.5 versus 8.7 minutes).

Although from a practical point of view all of these schemes are explicit and so are much better (faster, accurate) than the implicit, or semi-implicit schemes often used for these equations. This must be the main message to be taken from this article.

In figure (7) a 2-D comparison is performed. As well as the previous schemes, we also employed the Cash-Karp version of the RK4 scheme that adapts the timestep (the overhead is small, easily allowing this). It is, again, clear that the ETD4RK4 and ETD4RK4-B schemes are more accurate and over longer times ETD4RK4-B is the preferred scheme. The adaptive method is very fast, and the accuracy can be improved by lessening the error tolerances, and thus it is recommended for longer, more time-consuming, computations.

4. CONCLUDING REMARKS

We have developed and packaged a suite of algorithms for solving reaction diffusion equations. To make the algorithms immediately relevant and directly usable for those in the reaction diffusion equations community we have illustrated the algorithms upon recent and varied examples from the literature. Probably the most striking feature to emerge is how splendidly the method copes with sharp variations in the solutions, and also how fast and accurate the method is even with large timesteps.

But, before further congratulating ourselves upon the efficiency of spectral methods we must discuss several disadvantages:

The scheme we present is utterly reliant upon the reaction-diffusion equations being semi-linear, that is, the diffusion terms are simply $u_{xx} + Cu_{yy}$ and similarly for v ; for some constant C (we have simply had isotropic diffusion in this article).

We have not discussed possible problems with aliasing, earlier versions of our code utilized Orzag's 2/3 rule to filter this out. However, this actually made no

discernable difference to the solutions and we later just discarded this. It is evident that the method has terms $\exp(-\Omega^2 \Delta t)$ so higher order modes are, in any case, exponentially decaying; aliasing transfers some lower order modes to higher ones, so for diffusion-like problems the aliasing is automatically damped. Nonetheless aliasing is an issue that should be borne in mind in any spectral scheme.

Fourier spectral methods require periodicity, and we are not in the position, at least here, to set Neumann or Dirichlet boundary conditions on the edge of the domain. That requires an extension to Chebyshev, or some other basis functions. Thus we have to take the domain size large enough that the waves, pulses, structures of interest do not interact with the edges of the domain. In fact, one can set up and indeed solve Dirichlet/ Neumann boundary condition problems using integrating factor methods, see for instance [Kassam and Trefethen 2003], but there is then an essential difference. One must take the exponential of a full matrix, the periodic case treated here is special as those matrices are then diagonal and this simplification underlies all that we have done here, and computing the exponential of a matrix is numerically expensive particularly if it must be re-computed. This is an area that deserves further thought and work as the prospective pay-off in generating explicit timestepping codes for stiff PDEs in high spatial dimensions is considerable.

In some cases spectral accuracy means that we can use so few modes that the graphical solutions look unnaturally poor. This is despite the isolated values at the grid points being spectrally accurate, we can then utilize interpolation onto a finer grid using periodic cardinal functions, an algorithm for 1D is supplied in [Weideman and Reddy 2001]. We present an alternative, and generalization to 2D, in Appendix B based upon padding a Fourier transform with zeros.

Note that we are not claiming that the codes herein are the absolute best algorithms available for reaction-diffusion equations, nor do we attempt to imply that other scientists using alternative algorithms have been misguided. In particular, in 1D, the adaptive scheme of [Blom and Zegeling 1994] has proved itself to be a useful and accurate algorithm that we have enjoyed working with. One alternative scheme that certainly suggests itself is an Alternating Direction Implicit scheme where spatial discretization is again done through spectral methods, for completeness we provide a Matlab code that does this and we discuss this further in an appendix. In essence, we find that the low-order time solver usually used means that the scheme performs much less well than the integrating factor method of the main text.

Our aim has been, and is, to provide good, clear, working, versatile spectral schemes, that avoid stiffness issues, in a form whereby they can be utilized and built upon by other scientists. Thus, we hope, allowing them to concentrate upon the physics, biology, chemistry or other scientific issue rather than upon numerical concerns; the codes are summarized in table I, and are documented both internally and via an electronic README file.

Table I. A schematic tree of the provided algorithms. Names and corresponding equations are matched in the lower table.

Fortran	Matlab
-- OneD	-- OneD
-- CK45	-- CK45
-- auto_CK45.f	-- auto_CK45.m
-- epidemic_CK45.f	-- epidemic_CK45.m
-- fisher_CK45.f	-- fisher1D_CK45.m
'-- gray1D_CK45.f	'-- gray1D_CK45.m
-- ETD RK4_B	-- ETD RK4
-- auto_ETD RK4_B.f	'-- gray1D_ETD RK4.m
-- epidemic_ETD RK4_B.f	-- ETD RK4_B
-- fisher_ETD RK4_B.f	-- auto_ETD RK4_B.m
'-- gray1D_ETD RK4_B.f	-- epidemic_ETD RK4_B.m
'-- RK4	-- fisher1D_ETD RK4_B.m
-- auto_RK4.f	'-- gray1D_ETD RK4_B.m
-- epidemic_RK4.f	'-- RK4
-- fisher_RK4.f	-- auto_RK4.m
'-- gray1D_RK4.f	-- epidemic_RK4.m
'-- TwoD	-- fisher1D_RK4.m
-- CK45	'-- gray1D_RK4.m
-- gray2D_CK45.f	'-- TwoD
'-- labyrinthe2D_CK45.f	-- CK45
-- ETD RK4_B	-- fisher2D_CK45.m
-- gray2D_ETD RK4_B.f	-- gray2D_CK45.m
'-- labyrinthe2D_ETD RK4_B.f	'-- labyrinthe2D_CK45.m
'-- RK4	-- ETD RK4
-- gray2D_RK4.f	'-- labyrinthe2D_ETD RK4.m
'-- labyrinthe2D_RK4.f	-- ETD RK4_B
Useful	-- fisher2D_ETD RK4_B.m
-- adifisher.m	-- gray2D_ETD RK4_B.m
-- fourierupsample.m	'-- labyrinthe2D_ETD RK4_B.m
-- fourierupsample2D.m	'-- RK4
-- plot_fisher2D.m	-- fisher2D_RK4.m
-- plot_gray2D.m	-- gray2D_RK4.m
-- plot_labyrinthe2D.m	'-- labyrinthe2D_RK4.m

	File Name	ref. Equation
1-D	auto	(16)
	epidemic	(2)
	fisher1D	(13)
	gray1D	(15)
2-D	fisher2D	Appendix C
	gray2D	(18)
	labyrinthe2D	(19)

APPENDIX

A. THE ONE DIMENSIONAL MATLAB CODE

```

function gray1D_RK4(N,Nfinal,dt,ckeeep,L,epsilon,a,b)
if nargin<8;
    disp('Using default parameters');
    N=512; Nfinal=10000; dt=0.2; ckeeep=10;
    L=50; epsilon=0.01; a=9*epsilon; b=0.4*epsilon^(1/3);
end

x=(2*L/N)*(-N/2:N/2-1)';
u=initial(x,L); uhat=fft(u);
ukeep=zeros(N,2,1+Nfinal/ckeeep);
ukeep(:,:,1)=u;
tkeep=dt*[0:ckeeep:Nfinal];
ksq=((pi/L)*[0:N/2 -N/2+1:-1]')^2;
%-----Runge-Kutta-----
E=[exp(-dt*ksq/2) exp(-epsilon*dt*ksq/2)]; E2=E.^2;
for n=1:Nfinal
    k1=dt*fft(rhside(u,a,b));
    u2=real(ifft(E.*(uhat+k1/2)));
    k2=dt*fft(rhside(u2,a,b));
    u3=real(ifft(E.*uhat+k2/2));
    k3=dt*fft(rhside(u3,a,b));
    u4=real(ifft(E2.*uhat+E.*k3));
    k4=dt*fft(rhside(u4,a,b));
    uhat=E2.*uhat+(E2.*k1+2*E.*(k2+k3)+k4)/6;
    u=real(ifft(uhat));
    if mod(n,ckeeep)==0,
        ukeep(:,:,1+n/ckeeep)=u;
    end
end
save('gray1D_RK4.mat','tkeep','ukeep','N','L','x')
%-----Figures-----
mesh(tkeep,x,squeeze(ukeep(:,2,:))); view([60,75]);
xlabel('t'); ylabel('x'); zlabel('z');
title('(a) Surface plot of v')
%-----Initial Condition -----
function u=initial(x,L)
u=[1-0.5*(sin(pi*(x-L)/(2*L)).^100) ...
    0.25*(sin(pi*(x-L)/(2*L)).^100)];
%-----Right Hand Side-----
function rhs2=rhside(u,a,b)
t1=u(:,1).*u(:,2).*u(:,2);
rhs2=[-t1+a*(1-u(:,1)) t1-b*u(:,2)];

```

This produces figure 2(a) of the text for the Gray-Scott equations.

B. FOURIER CARDINAL FUNCTION INTERPOLATION

As noted in the text, spectral methods are often very accurate even with few interpolation points. When plotting graphically this sometimes leads to artificially poor-looking output, clearly the solution is spectrally accurate at each interpolation point and we just need to insert more points. Fourier cardinal function interpolation as in [Weideman and Reddy 2001] can be used in 1D, or, more in tune with the current article, one can pad an FFT with additional zeros and then invert which is convenient in either one or two space dimensions. The short Matlab scripts that do this are:

In one dimension:

```
function fout=fourierupsample(fin,newN);
% Given a periodic function fin, computed at N equispaced nodes in
% the periodic domain [-L,L], fout is its upsampled version on newN
% nodes onto the same domain.

N=length(fin); HiF=(N-mod(N,2))/2+1;
fftfin=max((newN/N),1)*fft(fin);
fout=real(ifft([fftfin(1:HiF); zeros(newN-N,1); fftfin(HiF+1:N)]));
```

And in two dimensions:

```
function fout=fourierupsample2D(fin,newNx,newNy);
% Given a periodic function fin in 2D computed at equidistant
% nodes Nx x Ny, then fout is its upsampled version on
% newNx x newNy nodes.

[Ny,Nx]=size(fin);
HiFx=(Nx-mod(Nx,2))/2+1; HiFy=(Ny-mod(Ny,2))/2+1;
fftfin=max((newNx/Nx)*(newNy/Ny),1)*fft2(fin);
fout=real(ifft2([fftfin(1:HiFy,1:HiFx), ...
    zeros(HiFy,newNx-Nx), fftfin(1:HiFy,HiFx+1:end); ...
    zeros(newNy-Ny,newNx); fftfin(HiFy+1:end,1:HiFx), ...
    zeros(Ny-HiFy,newNx-Nx), fftfin(HiFy+1:end,HiFx+1:end)]));
```

C. ALTERNATING DIRECTION IMPLICIT (ADI) METHODS:

This appears to be a viable alternative to that which we have presented in the main text; it is worth briefly outlining the method. The basic idea is similar to operator (Strang) splitting and the method is discussed in some detail in [Boyd 2001; Press et al. 1992]. As noted by Boyd the conventional centered finite difference schemes can easily be modified by using spectral differentiation matrices, let D denote the $N \times N$ Fourier differentiation matrix ([Fornberg 1998], [Boyd 2001], [Trefethen 2000], [Weideman and Reddy 2001]). ADI, or at least the ADI we use here, means that we split each time step into two and we first deal implicitly with one set of space derivatives and then in the next half time step with the other. So

$$u_t = u_{xx} + u_{yy} + f(u)$$

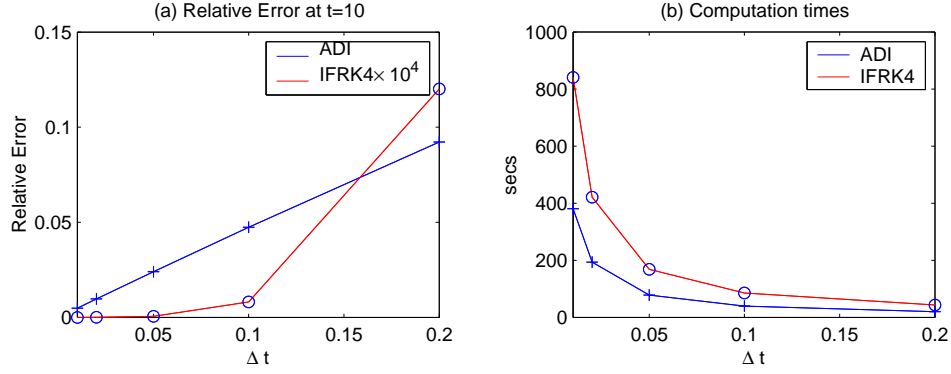


Fig. 8. Panels (a) and (b) relative errors and timings for ADI versus the integrating factor method.

is approximated by the following matrix system, here I is the $N \times N$ identity matrix and U is the matrix of u

$$U^{(t+1/2)} = \left[I - \frac{\Delta t}{2} D \right]^{-1} U^{(t)} \left[I + \frac{\Delta t}{2} D^T \right] + \left[I - \frac{\Delta t}{2} D \right]^{-1} \frac{\Delta t}{2} F(u^{(t)})$$

$$U^{(t+1)} = \left[I + \frac{\Delta t}{2} D \right] U^{(t+1/2)} \left[I - \frac{\Delta t}{2} D^T \right]^{-1} + \frac{\Delta t}{2} F(u^{(t+1/2)}) \left[I - \frac{\Delta t}{2} D^T \right]^{-1}$$

with the evident advantage that each matrix is evaluated only once and thereafter we just have matrix multiplication. This is ideal for implementation in Matlab. Unfortunately, this is only accurate to $O(\Delta t)^2$ so although the method is nicely stable one requires relatively small time steps relative to the explicit non-stiff scheme that is used in the main text. For instance, for Fisher's equation in 2D we show some comparative errors and timings in figure 8; note this simulation is for Fisher's equation using 256×256 Fourier modes on a 50×50 domain, the initial condition is a Gaussian $0.2 \exp(-0.25(x^2 + y^2))$. The integrating factor solution with $\Delta t = 10^{-2}$ is taken as the reference solution and the relative errors are computed as the maximal difference away from this. Notably the errors from the integrating factor scheme are multiplied by 10^4 in order that they are visible.

Doubtless one could improve the naive implementation above, but for the application to reaction-diffusion equations it seems uncompetitive. There are advantages though, in that it is generalizable to problems with u dependent diffusivity whereas the integrating factor method is not.

REFERENCES

- BALMFORTH, N. J. AND CRASTER, R. V. 2000. Dynamics of cooling domes of viscoplastic fluid. *J. Fluid Mech.* **422**, 225–247.
- BALMFORTH, N. J., CRASTER, R. V., AND MALHAM, S. J. A. 1999. Unsteady fronts in an autocatalytic system. *Proc. R. Soc. Lond. A* **455**, 1401–1433.
- BALMFORTH, N. J., CRASTER, R. V., AND SASSI, R. 2004. Dynamics of cooling viscoplastic domes II. *J. Fluid Mech.* **499**, 149–182.
- BAYLISS, A., MATKOWSKY, B. J., AND MINKOFF, M. 1989. Period doubling gained, period doubling lost. *SIAM J. Appl. Maths* **49**, 1047–1063.

Technical Report. Note del Polo, No. 99, 2006.

- BLOM, J. G. AND ZEGELING, P. A. 1994. Algorithm 731: a moving-grid interface for systems of one-dimensional time-dependent partial differential equations. *ACM Trans. Math. Software* 20, 194–214.
- BOYD, J. P. 2001. *Chebyshev and Fourier spectral methods*. Dover, New York.
- CAI, W. AND ZHANG, W. 1998. An adaptive spline wavelet ADI (SW-ADI) method for two-dimensional reaction-diffusion equations. *J. Comp. Phys.* 139, 92–126.
- CANUTO, C., HUSSAINI, M. Y., QUARTERONI, A., AND ZANG, T. 1988. *Spectral methods in fluid mechanics*. Springer-Verlag, New York.
- CASH, J. R. AND KARP, A. H. 1990. A variable-order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software* 16, 201–222.
- COX, S. M. AND MATTHEWS, P. C. 2002. Exponential time differencing for stiff systems. *J. Comp. Phys.* 176, 430–455.
- CRASTER, R. V. AND MATAR, O. K. 2000. Surfactant transport on mucus films. *J. Fluid Mech.* 425, 235–258.
- DAVIDSON, F. A., SLEEMAN, B. D., RAYNER, A. D. M., CRAWFORD, J. W., AND RITZ, K. 1997. Travelling waves and pattern formation in a model for fungal development. *J. Math. Biol.* 35, 589–608.
- DOELMAN, A., KAPER, T. J., AND ZEGELING, P. A. 1997. Pattern formation in the one-dimensional Gray-Scott model. *Nonlinearity* 10, 523–563.
- DORMAND, J. R. AND PRINCE, P. J. 1980. A family of embedded runge-kutta formulae. *J. Comp. and Appl. Math.* 6, 19–26.
- FISHER, R. A. 1937. The wave of advance of advantageous genes. *Ann. Eugenics* 7, 353–369.
- FORNBERG, B. 1998. *A practical guide to Pseudospectral methods*. Cambridge University Press, Cambridge.
- HAGBERG, A. AND MERON, E. 1994. From Labyrinthine patterns to spiral turbulence. *Phys. Rev. Lett.* 72, 2494–2497.
- HIGHAM, N. J. 1996. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia.
- JONES, W. B. AND O'BRIEN, J. J. 1996. Pseudo-spectral methods and linear instabilities in reaction-diffusion fronts. *Chaos* 6, 219–228.
- KASSAM, A. AND TREFETHEN, L. N. 2003. Fourth-order time stepping for stiff PDEs. <http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/etd.ps.gz>.
- KEAST, P. AND MUIR, P. H. 1991. Algorithm 688 EPDCOL - a more efficient PDECOL code. *ACM Trans. Math. Software* 17, 153–166.
- KROGSTAD, S. 2005. Generalized integrating factor methods for stiff PDEs. *Journal of Computational Physics* 203, 72–88.
- LI, S. T., PETZOLD, L., AND REN, Y. H. 1998. Stability of moving mesh systems of partial differential equations. *SIAM J. Sci. Comp.* 20, 719–738.
- LIAO, W., ZHU, J., AND KHALIQ, A. Q. M. 2002. An efficient high-order algorithm for solving systems of reaction-diffusion equations. *Numer. Methods Partial Differential Eq.* 18, 340–354.
- MELGAARD, D. K. AND SINCOVEC, R. F. 1981. General software for two-dimensional nonlinear partial differential equations. *ACM Transactions on Mathematical Software* 7, 107–125.
- MERON, E., BÄR, M., HAGBERG, A., AND THIELE, U. 2001. Front dynamics in catalytic surface reactions. *Catalysis Today* 70, 331–340.
- METCALF, M. J., MERKIN, J. H., AND SCOTT, S. K. 1994. Oscillating wave fronts in isothermal chemical systems with arbitrary powers of autocatalysis. *Proc. Roy. Soc. Lond. A* 447, 155–174.
- MUNTKE-KAAS, H. 1999. High order Runge-Kutta methods on manifolds. *Applied Numerical Mathematics* 29, 115–127.
- MURATOV, C. B. AND OSIPOV, V. V. 2001. Spike autosolitons and pattern formation scenarios in the two-dimensional Gray-Scott model. *Eur. Phys. J. B.* 22, 213–221.
- MURATOV, C. B. AND OSIPOV, V. V. 2002. Stability of the static spike autosolitons in the two-dimensional Gray-Scott model. *SIAM J. Appl. Math.* 62, 1463–1487.
- MURRAY, J. D. 1993. *Mathematical Biology, 2nd Edition*. Springer-Verlag, New York.
- Technical Report. Note del Polo, No. 99, 2006.

- PEARSON, J. E. 1993. Complex patterns in a simple system. *Science* 261, 189–192.
- PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 1992. *Numerical Recipes: The art of scientific computing*. Cambridge University Press, Cambridge.
- QIU, Y. AND SLOAN, D. M. 1998. Numerical solution of Fisher’s equation using a moving mesh method. *J. Comp. Phys.* 146, 726–746.
- RAMOS, J. I. 2002. Wave propagation and suppression in excitable media with holes and external forcing. *Chaos, Solitons and Fractals* 13, 1243–1251.
- REYNOLDS, W. N., PONCE-DAWSON, S., AND PEARSON, J. E. 1997. Self-replicating spots in reaction–diffusion systems. *Phys. Rev. E* 56, 185–198.
- SHERRATT, J. A. 1997. A comparison of two numerical methods for oscillatory reaction-diffusion systems. *Appl. Math. Lett.* 10, 1–5.
- SKEEL, R. D. AND BERZINS, M. 1990. A method for the spatial discretization of parabolic equations in one space variable. *SIAM J. Sci. Stat. Comput.* 11, 1, 1–32.
- TANG, S., QIN, S., AND WEBER, R. O. 1993. Numerical studies on 2-dimensional reaction-diffusion equations. *J. Aust. Math. Soc. Ser. B* 35, 223–243.
- TREFETHEN, L. N. 2000. *Spectral methods in Matlab*. SIAM Publications, Philadelphia.
- VANDEN BERGHE, G., DE MEYER, H., VAN DAELE, M., AND VAN HECKE, T. 2000. Exponentially fitted Runge-Kutta methods. *J. Comp. Appl. Math.* 125, 107–115.
- WEIDEMAN, J. A. C. AND REDDY, S. C. 2001. A MATLAB differentiation matrix suite. *ACM Transactions on Mathematical Software* 26, 465–519.